BIN-PACKING

Вандакуров Артем

22 января 2023 г.

Даны числа $a_1, ..., a_n \in (0, 1]$. Требуется разбить их на минимальное число групп, так чтобы сумма чисел в каждой группе не превышала 1

1 Для любого ϵ>0 задача о ($\frac{3}{2}$-ϵ)-приближении является NP-трудной.

Мы сводим от Partition, который, как мы знаем, является NP-полным. В Partition нам даны n чисел c_1, . . . , $c_n \in$ N и просят узнать, существует ли множество S $\subseteq$ {1, . . . , n} такое, что $\sum_{i\in S} c_i = \sum_{i\notin S} c_i$. По данному элементу из Partition мы создаем элемент для BIN-PACKING, устанавливая $s_i = 2c_i/(\sum_{j=1}^{n} c_j) \in (0, 1], 1 \leq i \leq n$. Очевидно, двух групп достаточно тогда и только тогда, когда существует S $\subseteq${1, . . . , n} такое, что $\sum_{i\in S} c_i = \sum_{i\notin S} c_i$. Это позволяет нам получить нижнюю границу аппроксимируемости Bin Packing

2 Для любого $\epsilon > 0$ постройте алгоритм, дающий $(1 + \epsilon)$OPT + 1 корзин для оптимального значения OPT.

Покажем, что существует алгоритм, дающий $(1 + \epsilon)$OPT + 1 корзин для оптимального значения OPT.

Теорема. Для любого $0 < \epsilon \leq 1/2$ существует алгоритм, работающий за полиномиальное время от n и находит решение, имеющее не более k (1 + ϵ)OPT + 1 групп.

Лемма 1. Пусть $\epsilon > 0$ и d $\in$ N — константы. Для любой задачи BIN-PACKING, где $s_i \geq \epsilon$ и n $\leq$ d, существует алгоритм с полиномиальным временем, который решает ее оптимально.

Доказательство. Количество предметов в корзине ограничено m := $\lfloor 1/\epsilon \rfloor$. Следовательно, число различных разбиений для одной группы ограничено r = $\frac{(m+d)!}{m!d!}$, что является константой. Используется не более n групп, поэтому количество допустимых назначений ограничено p = $\frac{(n+r)!}{n!r!}$. Это многочлен от n. Таким образом, мы можем перебрать все варианты и выбрать лучший из них, чтобы дать оптимальное решение.

Доказано.

Лемма 2. Пусть $\epsilon > 0$ — константа.Для любой задачи BIN-PACKING, где $s_i \geq \epsilon$, существует алгоритм $(1 + \epsilon)$-аппроксимации.

Доказательство. Пусть I будет данной задачей. Отсортируем n элементов по возрастанию и разделим их в g = $\lfloor 1/\epsilon^2 \rfloor$ групп, каждая из которых имеет не более q = $\lfloor n\epsilon^2 \rfloor$ предметов. Заметим, что две группы могут быть одинакового размера. Создадим J := I. Далее присваиваем каждому элементу J наибольший элемент из его группы. J имеет не более g различных элементов. Поэтому мы можем найти оптимальное назначение для J, применив лемму 1. Теперь покажем, что OPT(J) $\leq$ $(1+\epsilon)$ OPT(I): мы создаем K := I. Далее присваиваем каждому элементу K наименьший элемент из его группы. Тогда OPT(K) $\leq$ OPT(J). Также заметим, что OPT(J) $\leq$ OPT(K) + q $\leq$ OPT(I) + q. Поскольку каждый элемент не менее ϵ, мы имеем OPT(I) $\geq$ nϵ и q = $\lfloor n\epsilon^2 \rfloor \leq \epsilon$OPT(I). Следовательно, OPT(J) $\leq$ $(1 + \epsilon)$OPT(I).

Доказано.

Доказательство теоремы. Пусть I - данный набор и J - набор после удаления элементов, меньших ϵ из I. Мы можем применить лемму 2 и найти разбиение, которое использует не более k(J) $\leq$ $(1 + \epsilon)$k * OPT(I) групп. Сначала, мы включаем элементы, меньшие ϵ в решение, найденное в J. Используем дополнительные группы если элемент не помещается ни в одну из используемых до сих пор. Если дополнительные группы не нужны, то наше решение использует k(I) $\leq$ $(1 + \epsilon) \cdot$ OPT(J) $\leq$ $(1 + \epsilon)$OPT(I) групп. В противном случае остаточная вместимость всех групп, кроме последнего, меньше ϵ. Таким образом, s(I) $\geq$ (k(I) - 1)(1 - ϵ), что является оценкой снизу для OPT(I). Таким образом, к(I) $\leq \frac{OPT(I)}{1-\epsilon}$ + 1 $\leq$ $(1 + 2\epsilon)$OPT(I) + 1, где мы использовали $0 < \epsilon \leq 1/2$.

Доказано.

3 Реализация алгоритма

```
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

int n = 0;
double K = 1; // maximum sum of elements in one group
double min_size; // minimum element
double max_size;// maximum element
double sum; // sum of all elements
```

```
vector <double> load_data()
{
        cin >> n;
        vector <double> elements(n);

        min_size = K;
        max_size = 0;

        int i = 0;
        while (i < n) {
                cin >> elements[i];
                min_size = min(min_size, elements[i]);
                max_size = min(max_size, elements[i]);
                sum += elements[i];
                ++i;
        }
        return elements;
}


unsigned int find_partition(vector <double> elements)
{
        int num_open_bins = 1;
        int num_full_bins = 0;

        priority_queue<double, vector<double>, greater<double > > pq;
        pq.push(0);

        double limit_capacity = K - min_size;
        double bin = 0;

        for (int i = 0; i < n; i++)
        {
                if (pq.empty())
                        bin = K;
                else
                        bin = pq.top();

                if (bin + elements[i] <= K)
                {
                        bin += elements[i];
                        pq.pop();       // delete group; if its size is signif
                        
                        if (bin > limit_capacity)
                        {
```

```
                                num_open_bins--;
                                num_full_bins++;
                        }
                        else
                                pq.push(bin);
                }

                // create new group
                else
                {
                        num_open_bins++;
                        pq.push(elements[i]);
                }
        }

        return num_open_bins + num_full_bins;
}

int main()
{
        vector <double> elements = load_data();
        cout << find_partition(elements);
}
```